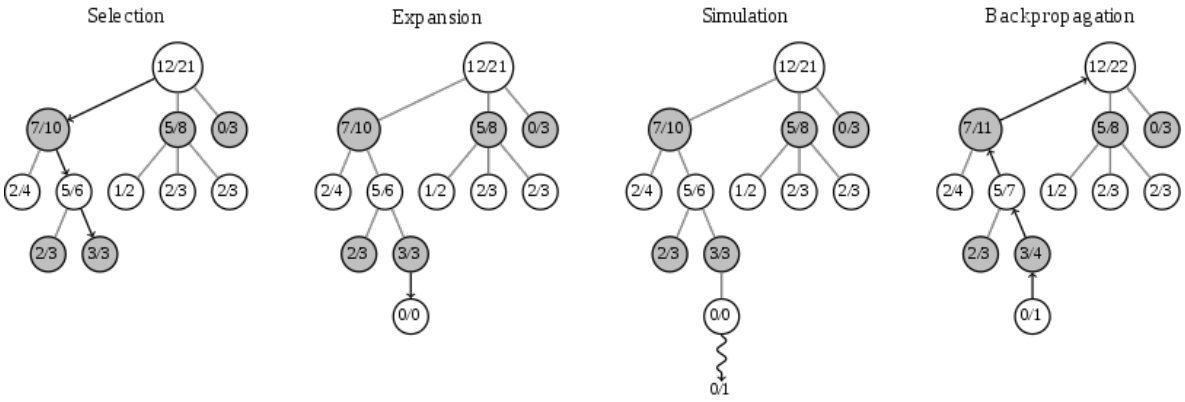
Search Algorithms of AlphaGo Fan and AlphaGo Zero

Seok Won Lee

ijleesw@gmail.com

1 Monte Carlo Tree Search (MCTS)



`http://mongxmongx2.tistory.com/17`, `https://spin.atomicobject.com/2015/12/12/monte-carlo-tree-search-algorithm-game-ai/` 참고.

2 Search Algorithm of AlphaGo Fan [1]

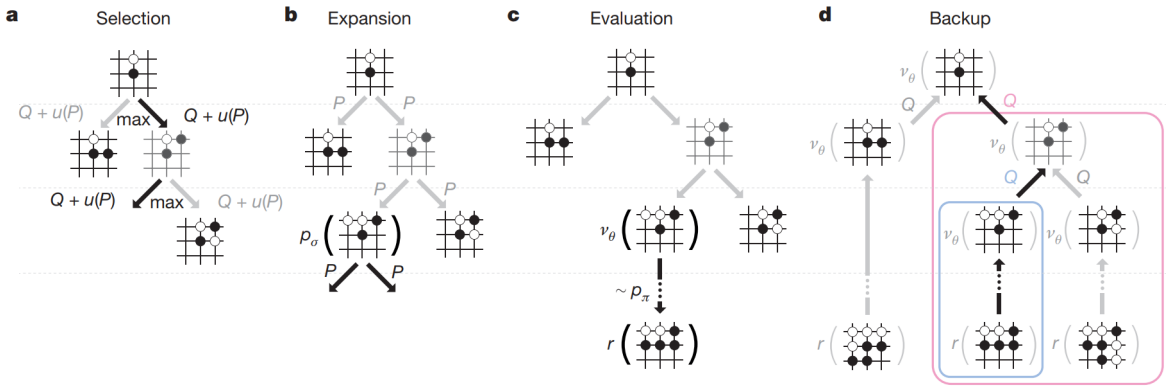
AlphaGo Fan은 main search를 담당하는 single master machine과 함께, rollout을 담당하는 remote CPUs, 그리고 policy and value network evaluation을 담당하는 remote GPUs로 이루어져 있다. Search tree는 master machine에 저장되며, remote CPUs는 rollout policy로 simulation을 하고, remote GPUs는 policy and value networks를 계산한다.

2.1 Notation 및 설명

- s : 현재 바둑판의 상태 (state)
- a : 바둑돌 두기 (action)

여기서 s는 directed graph의 node, a는 directed graph의 edge라고 생각하면 된다. 각 node s는 모든 action a에 대해 다음의 정보를 보관한다.

- $P(s,a)$: 사전 확률(Prior probability). $(= p_\sigma(s,a))$



- $N_r(s,a)$: Search algorithm이 해당 노드를 거쳐간 횟수의 총합. (= visit count)
- $W_r(s,a)$: 해당 노드를 거쳐서 leaf node에 도달한 뒤 시행된 rollout의 결과값 z_T의 총합.
- $N_v(s,a)$: Search algorithm이 해당 노드를 거쳐간 횟수의 총합. (= visit count)
- $W_v(s,a)$: 해당 노드를 거쳐서 leaf node에 도달한 뒤 계산된 value network의 결과값 $v_\theta(s_L)$의 총합.
- $Q(s,a)$: state s에서 action a를 취했을 때의 expected value.

2.2 Selection

현재 state $s = s_0$에서 출발하여, 각 time-step t에 대해 state s_t에서

$$a_t = \underset{a}{\text{argmax}}[Q(s_t,a) + u(s_t,a)], \quad u(s,a) = c_{puct} P(s,a) \frac{\sqrt{\sum_b N_r(s,b)}}{1 + N_r(s,a)} \propto \frac{P(s,a)}{1 + N(s,a)}$$

인 action a_t를 고른다. 이를 leaf node s_L을 만날 때까지 반복한다. (Experiment에서는 $c_{puct} = 5$로 설정함.)

$u(s,a)$를 위와 같이 정의함으로써 1) 초기에는 $P(s,a)$가 높고 visit count가 낮은 node를 선택하도록 유도하고, 2) asymptotically $Q(s,a)$가 높은 node를 선택하도록 한다.

2.3 Expansion

Visit count $N_r(s,a)$가 threshold n_{thr}보다 커지게 되면 search tree에 successor state $s' = f(s,a)$를 추가한다. 새 node s'는

$$N_r(s',a) = N_v(s',a) = 0, W_r(s',a) = W_v(s',a) = 0, P(s',a) = p_\sigma(a|s')$$

로 initialize 된다.

n_{thr}은, expansion이 시행되어 새로운 position이 policy network의 queue에 추가되는 속도와 remote GPUs가 policy network를 evaluate하는 속도가 일치하도록 dynamic하게 조정된다. (Experiment에서는 $n_{thr} = 40$으로 설정함.)

2.4 Evaluation

Leaf node s_L에 도착했으면 1) 해당 node를 value network의 queue에 넣어서 $v_\theta(s_L)$를 구하고, 2) remote CPUs의 rollout policy를 사용하여 진행한 시뮬레이션의 결과값 z_T를 구한다.

2.5 Backup

$t \leq L$일 때, 해당 thread가 방문한 node를 다른 thread가 지나치게 방문하는 걸 막기 위해, 매번 node를 방문할 때마다 다음과 같이 virtual loss n_{vl}을 더하거나 빼준다.

$$N_r(s_t, a_t) \leftarrow N_r(s_t, a_t) + n_{vl}$$
$$W_r(s_t, a_t) \leftarrow W_r(s_t, a_t) - n_{vl}$$

이후 Leaf node에 대한 evaluation이 끝나고 난 뒤에는 아래와 같이 update 해준다.

$$N_r(s_t, a_t) \leftarrow N_r(s_t, a_t) - n_{vl} + 1$$
$$W_r(s_t, a_t) \leftarrow W_r(s_t, a_t) + n_{vl} + z_t$$
$$N_v(s_t, a_t) \leftarrow N_v(s_t, a_t) + 1$$
$$W_v(s_t, a_t) \leftarrow W_v(s_t, a_t) + v_\theta(s_L)$$

결국 $N_r(s_t, a_t)$와 $N_v(s_t, a_t)$는 visit count이므로 1씩 증가하게 된다. $W_r(s_t, a_t)$에는 rollout policy의 결과를 더해주고, $W_v(s_t, a_t)$에는 value network의 결과를 더해주게 된다.

최종적으로, $Q(s, a) = (1 - \lambda)\frac{W_v(s_t,a_t)}{N_v(s_t,a_t)} + \lambda\frac{N_r(s_t,a_t)}{N_r(s_t,a_t)}$로 expected value를 update 해준다. (Experiment에서는 $\lambda = 0.5$로 설정함.)

3 Search Algorithm of AlphaGo Zero [2]

3.1 Notation 및 설명

AlphaGo Zero에서 search tree의 각 node s는 모든 action a에 대해 다음의 정보를 보관한다.

- $P(s, a)$: 사전 확률(Prior probability). (= $\boldsymbol{p}$ from $(\boldsymbol{p}, v) = f_\theta(s)$)
- $N(s, a)$: Search algorithm이 해당 노드를 거쳐간 횟수의 총합. (= visit count)
- $W(s, a)$: 해당 노드를 거쳐서 leaf node에 도달한 뒤 시행된 deep neural network f_θ의 결과값 v의 총합. (= total action value)
- $Q(s, a)$: state s에서 action a를 취했을 때의 expected value.

3.2 Selection

Leaf node s_L을 만날 때까지 $a_t = \text{argmax}_a[Q(s_t, a) + u(s_t, a)]$인 a_t를 선택한다. $u(s_t, a)$는 위와 동일하다.

3.3 Expansion and Evaluation

Leaf node s_L에 도달하면 먼저 s_L을 돌리거나 뒤집은 모습의 copy를 $\tilde{s_L}$에 저장한다. ($\tilde{s_L}$은 내 표기.) 그리고 $\tilde{s_L}$을 neural network evaluation queue에 넣어서 $(\tilde{\boldsymbol{p}}, v) = f_\theta(\tilde{s_L})$를 계산하고 v를 잠시 저장해둔다. (rollout policy를 쓰지 않는다!)

다음으로 leaf node s_L을 expand한다. (Zero에는 threshold가 없다.) 모든 action a에 대해,

$$N(s_L, a) = W(s_L, a) = 0, \quad Q(s_L, a) = 0, \quad P(s_L, a) = \Pr(a|s_L) \text{ from } \boldsymbol{p}$$

로 initialize한다.

AlphaGo Fan에서는 master machine의 thread가 leaf node에 도달하면 해당 node를 remote CPUs & GPUs에 넘기고 결과를 받기 전에 새로운 search를 시작하였다. 이와 달리 AlphaGo Zero는 thread가 leaf node에 도달하면 새로운 search를 시작하는 대신 결과값 v를 받을 때까지 계속 기다린다.

3.4 Backup

Evaluation과 Expansion이 끝나면 결과값 v를 가지고 다음과 같이 update를 한다.

$$\begin{aligned} N(s_t, a_t) &\leftarrow N(s_t, a_t) + 1 \\ W(s_t, a_t) &\leftarrow W(s_t, a_t) + v \\ Q(s_t, a_t) &\leftarrow \frac{W(s_t, a_t)}{N(s_t, a_t)} \end{aligned}$$

4 AlphaGo Zero와 AlphaGo Fan의 차이점

- Zero는 rollout을 쓰지 않는다.
- Fan은 policy and value networks를 쓰는 반면, Zero는 neural network를 하나만 쓴다. (f_θ)
- Fan은 leaf expansion에 threshold를 둔 반면, Zero는 항상 leaf expansion을 한다.
- Zero의 각 thread는 neural network evaluation의 결과값이 나올 때까지 기다린다.

References

1. D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.
2. D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.